ECE 375 LAB 7

Timers/Counters

Lab Time: Friday 4-6

Faaiq Waqar

Jordan Brown

Introduction

Lab 7 introduced us to the applications of the timers and counters found on the atmega 128. We were required to use the fast PWM to change the brightness of 2 LEDs and represent the brightness level using 4 other LEDs. PWM allows us to output a voltage between 0 and 5 volts by quickly turning on and off the LED in a controlled manner. The duty cycle, or amount of time the LED is on, controls the brightness of it. There are also 2 other buttons that will bring the LEDs to full brightness or bring them down to a duty cycle of 0. During these processes the 4 LEDs on the left will jump to represent the corresponding value. These routines require the correct configuration of I/O ports, interrupts, and timer counter 0 and 2. We have already had lots of experience with I/O and interrupts, but the timer configuration allows the function as fast PWM using the system clock in a non inverting mode. The onboard timers allow for a variety of counting methods and can be used in many different ways. The waveform generator can be used to create any waveform using the input from the counters.

Program Overview

The program is designed to display the motor status of the bot by using pwm to increase or decrease LED brightness. LEDs 6 and 5 are always on which shows the bot is constantly travelling forward. LEDs 7 and 4 are controlled through PWM to show the speed level of the motor. The rightmost LEDs display the binary value of the speed level which is 0-15. Inside of the main function we check for input from the buttons and call to corresponding function for each press. There are 2 functions that instantly put the bot into maximum speed or bring it to a halt. The other 2 routines increase or decrease the speed by 1 level. Each of these functions achieves this by varying the value of OCR for each counter. This increases or decreases the duty cycle by changing the maximum value the counter will reach.

Initialization Routine

First we begin by setting all of the I/O ports we will use to input or output, and all of the buttons used will be configured with a pull up resistor. Since our second variation of the code used the polling method there was no need to configure the EICRA and EIMSK. Next, we set both counters to use the fast PWM in a non-inverting mode. We also set CS00, CS01, CS02 so the counter would be using the 16MHz system clock. With all these things in place we are ready to begin using our other routines to control the LEDs and their brightnesses.

Main Routine

Since we decided to use polling our main function is responsible for reading each of the inputs from the 4 buttons and comparing the values to see if any were pressed. There are 4 checks inside of main, 1 check for each function we implemented. Once the value is read to mpr we compare it to check if we should make a call to the respective routine.

SLOW_DOWN Routine

To prevent the wrap around from the speed up or slow down buttons each of these functions begins with a check to see if we are at the minimum value. If so we break and return to the bottom of main. If this condition is not met then we continue on to the main body of the routine. Here we decrement our count, which represents 0-15, for

the LED display and send it to the display. Then OCR for each counter is reduced by 1/16th of its max value which will cut the brightness by roughly 6.3 percent.

SPEED_UP Routine

Speed up serves a similar purpose to slow down but instead of dimming the LEDs is increase the brightness by 6.3 percent. This is achieved by checking if we are at the max value of 255, which will prevent the wrap around of the brightness levels. Then OCR is changed for both counters so the overflow happens slightly later which results in a longer duty cycle and brighter LED

MAX_SPEED Routine

Once the input from this button becomes low MAX_SPEED changes OCR0 and OCR2 to 255 so the LEDS will be given the full 5 volt output. The count for the binary expression is also change so the 4 LEDs display the value 15.

MIN_SPEED

This functions very similarly to the max speed function. Once the corresponding button is pressed OCR0 and OCR2 are set to 0 which is the min value counter 0 and 2 can count to. This means they will be operating at a 0% duty cycle and the LEDs will be at minimum brightness which is essentially 0 volts. The 4 LEDs on the right of the board are also changed to display the binary value of 0.

Additional Questions

1. In this lab, you used the Fast PWM mode of both 8-bit Timer/Counters, which is only one of many possible ways to implement variable speed on a TekBot. Suppose instead that you used just one of the 8-bit Timer/Counters in Normal mode, and had it generates an interrupt for every overflow. In the overflow ISR, you manually toggled both Motor Enable pins of the TekBot, and wrote a new value into the Timer/Counter's register. (If you used the correct sequence of values, you would be manually performing PWM.) Give a detailed assessment (in 1-2 paragraphs) of the advantages and disadvantages of this new approach, in comparison to the PWM approach used in this lab.

We should start here with an understanding of what the Fast PWM mode allows us to do, and why it's important. What fast PWM allows us to do is generate a high frequency pulse or square waves. because we can set a pulse wave to vary the average values of the wave form, it is easy to create something that will perhaps change the voltage into average basis, which we implemented here in this lab. However, it is important to understand that this is not at all how the normal mode works. In normal mode, we will be working with the initial count, which works to be a load value for initialization that doesn't stay throughout, a max value, which is the max counter, and an overflow flag we can use to interrupt.

A very large disadvantage of the normal mode would be the need to reset that count every time to generate a utility waveform that we could use to consistently modify the voltage to the motors, which would prove to be very code intensive, and not nearly as practical. Creating a waveform here would prove to be more difficult, and thus likely not nearly as practical, as the consistency basis of the fast pwm will prove to be the strongest point of utility in this lab. On the more positive side, normal mode takes far less set up, which could prove to be easier when actually programming, but this is far overshadowed by the pwm increased capability to create the consistent frequency clock.

2. The previous question outlined a way of using a single 8-bit Timer/Counter in Normal mode to implement variable speed. How would you accomplish the same task (variable TekBot speed) using one or both of the 8- bit Timer/Counters in CTC mode? Provide a rough-draft sketch of the Timer/Counter-related parts of your design, using either a flow chart or some pseudocode (but not actual assembly code).

In order to create this, we would need to utilize the CTC modes different setups for the max value that we will be setting up. In that, we can create 15 different max modes, and then load those into the output compare registers used to set those max values for the CTC mode. Using this, we could create the different generations for the motors to output a specific voltage

Pseudocode:

Initialize Function:

- Initialize timer 0 and timer 2 (8 bit) to CTC mode
- Set the output compare register to 0
- Have the interrupt ready per overflow

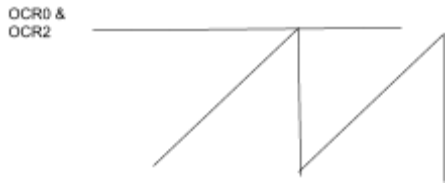
Main Function:

- Have the bot moving forward
- Listening for interrupt

Overflow and increment function:

- Increment OCR0 and OCR2 with +17, similar to PWM

Diagram:



Difficulties

Our understanding of this lab was very good, but despite this we were unable to get our program working using interrupts. After much frustration we decided to switch to the polling method and reimplement our functions that controlled the LEDs. We check our configurations a countless number of times and tried many different things for the functions that changed speed. With our new method were able to quickly recreate our program and after 30 minutes of debugging our program worked.

Conclusion

This lab was quite challenging but was a good test of skill using the counters. The counters are relatively easy to use and controlling the brightness of the 2 LEDs was the simplest part of the lab. The extensive usage of I/O was also good practice because it is such an important fundamental for programming with AVR. We were very

proficient with interrupts, but due to frustration we weren't able to see our mistake. This just allowed us to get more practice with AVR and the polling method.

Source Code

```
;***********************************************************
;*
;*      Faaiq_Waqar_and_Jordan_Brown_Lab7_sourcecode.asm
;*
;*      We use big timer
;*
;*      This is the skeleton file for Lab 7 of ECE 375
;*
;***********************************************************
;*
;*       Author: Faaiq Waqar
;*         Date: November 15th, 2019
;*
;***********************************************************


.include "m128def.inc"                      ; Include definition file


;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                           ; Multipurpose register
.def    str = r17


.equ    WskrR = 0                           ; Right Whisker Input Bit
.equ    WskrL = 1                           ; Left Whisker Input Bit


.equ    EngEnR = 4                          ; Right Engine Enable Bit
```

```
.equ    EngEnL = 7                          ; Left Engine Enable Bit
.equ    EngDirR = 5                         ; Right Engine Direction Bit
.equ    EngDirL = 6                         ; Left Engine Direction Bit

.equ    MovFwd = (1<<EngDirR|1<<EngDirL)    ; Move Forward Command
.equ    MovBck = $00                        ; Move Backward Command
.equ    TurnR = (1<<EngDirL)                ; Turn Right Command
.equ    TurnL = (1<<EngDirR)                ; Turn Left Command
.equ    Halt = (1<<EngEnR|1<<EngEnL)        ; Halt Command
.equ    Step = 17;

;***********************************************************
;*      Start of Code Segment
;***********************************************************
.cseg                                               ; beginning of code segment

;***********************************************************
;*      Interrupt Vectors
;***********************************************************
.org    $0000
               rjmp    INIT                 ; reset interrupt

               ; place instructions in interrupt vectors here, if needed

.org    $0046                               ; end of interrupt vectors

;***********************************************************
;*      Program Initialization
;***********************************************************
INIT:
```

```
; Initialize the Stack Pointer
ldi r16, high(RAMEND) ; Prepare lower stack addr
out SPH, r16                ; Store lower stack addr
ldi r17, low(RAMEND)  ; Prepare upper stack addr
out SPL, r17                ; Store upper stack addr
; Initialize the equidistant speed levels

; Configure I/O ports
ldi         mpr, $FF            ; Set Port B Data Direction Register
out         DDRB, mpr           ; for output
ldi         mpr, $00            ; Initialize Port D Data Register
out         PORTD, mpr          ; so all Port D inputs are Tri-State

ldi         mpr, (1<<4|1<<5|1<<6|1<<7)          ; Set Port B Data Direction
Register
out         DDRD, mpr           ; for output
ldi         mpr, (1<<0|1<<1|1<<2|1<<3)          ; Initialize Port D Data
Register
out         PORTD, mpr          ; so all Port D inputs are Tri-State
; Configure External Interrupts, if needed

; Configure 8-bit Timer/Counters
ldi mpr, (1<<WGM01|1<<WGM00|1<<COM01|1<<CS00)
out TCCR0, mpr

ldi mpr, (1<<WGM21|1<<WGM20|1<<COM21|1<<CS20)
out TCCR2, mpr
; no prescaling

ldi         r23, 0
ldi         mpr, MovFwd         ; Move the robot forward infiniely
```

```
                out             PORTB, mpr

                ldi             mpr, 0b00000100             ; Set up wait timer
                out             TCCR1B, mpr
                ; Set TekBot to Move Forward (1<<EngDirR|1<<EngDirL)
                ; Enable global interrupts (if any are used)

                ; Configure the External Interrupt Mask

                ; Enable Interrupts to be used in program
                sei



;***********************************************************
;*      Main Program
;***********************************************************
MAIN:
                in              r20, PIND
                cpi             r20, 0b00000111             ; Check for Button 0 Input
                breq    SPEED_MIN_CALL        ; if pressed, branch to function minimum
                cpi             r20, 0b00001011             ; Check for Button 1 Input
                breq    SPEED_MAX_CALL        ; If pressed, branch to function ma
                cpi             r20, 0b00001110             ; Check for Button 2 Input
                breq    SPEED_UP_CALL         ; if pressed, branch to function pd0
                cpi             r20, 0b00001101             ; Check for Button 3 Input
                breq    SPEED_DOWN_CALL              ; If pressed, branch to function pd1
                rjmp    MAIN                  ; Create an infinite while loop to signify the
                ; poll Port D pushbuttons (if needed)
SPEED_MIN_CALL:
```

```
                ldi             r23, 0                          ; utilize thr counter we use and set
to 0
                mov             mpr, r23                        ; copy the contents of the copy reg
to mpr
                ori             mpr, 0b01100000                 ;  Use  or  functionality  to  combine
the two for port
                out             PORTB, mpr                      ;  place  the  contentsin  portb  for
leds
                rcall   SPEED_MIN
SPEED_MIN_WAIT:
                in              r20, PIND
                cpi             r20, 0b00001111                 ; Check for button up Input, wait
                breq    JUMP_MAIN
                rjmp    SPEED_MIN_WAIT
SPEED_MAX_CALL:
                ldi             r23, 15                         ; utilize thr counter we use and set
to 15
                mov             mpr, r23                        ; copy the contents of the copy reg
to mpr
                ori             mpr, 0b01100000                 ;  Use  or  functionality  to  combine
the two for port
                out             PORTB, mpr                      ;  place  the  contentsin  portb  for
leds
                rcall   SPEED_MAX                       ; Jump to function to modify PWM
SPEED_MAX_WAIT:
                in              r20, PIND
                cpi             r20, 0b00001111                 ; Check for button up Input, wait
                breq    JUMP_MAIN
                rjmp    SPEED_MAX_WAIT
SPEED_UP_CALL:
                nop
                rcall   WAIT_2
                ldi             mpr, 15                         ; load multipurpose resiter to max
15
```

```
                cp              mpr, r23                    ; Compare to check if counter is at
a max
                breq    JUMP_MAIN                   ; branch off if the counter will overflow


                inc             r23                             ; Increment the counter
                mov             mpr, r23                    ; Copy the counter into mpr
                ori             mpr, 0b01100000             ;  Use  the  OR  command  function  to
combine
                out             PORTB, mpr                  ; output onto port B
                rcall   SPEED_DOWN
SPEED_UP_WAIT:
                in              r20, PIND
                cpi             r20, 0b00001111             ; Check for button up Input, wait
                breq    JUMP_MAIN
                rjmp    SPEED_UP_WAIT
SPEED_DOWN_CALL:
                nop
                rcall   WAIT_2
                ldi             mpr, 0                      ; load multipurpose resiter to min 0
                cp              mpr, r23                    ; Compare to check if counter is at
a min
                breq    JUMP_MAIN                   ; branch off if the counter will overflow


                dec             r23                             ; decrement the counter
                mov             mpr, r23                    ; Copy the counter into mpr
                ori             mpr, 0b01100000             ;  Use  the  OR  command  function  to
combine
                out             PORTB, mpr                  ; output onto port B
                rcall   SPEED_UP
SPEED_DOWN_WAIT:
                in              r20, PIND
                cpi             r20, 0b00001111             ; Check for Button 0 Input
```

```
                breq    JUMP_MAIN
                rjmp    SPEED_DOWN_WAIT
JUMP_MAIN:
                rjmp    MAIN
                                                        ; if pressed, adjust speed
                                                        ; also, adjust speed indication

;***********************************************************
;*      Functions and Subroutines
;***********************************************************

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
SPEED_DOWN:     ; Begin a function with a label

                in              mpr, OCR0               ; take the value currently stores in output
compare red
                ldi             str, 17                 ; Load to prepare a timer counter increment
                add             mpr, str                ; add to the loaded value

                out             OCR0, mpr               ; Place for timer 0
                out             OCR2, mpr               ; Place for timer 2



                ret
```

```
SPEED_UP:

              in            mpr, OCR0            ; take the value currently stores in output
compare red
              ldi           str, 17              ; Load to prepare a timer counter increment
              sub           mpr, str             ; subtract to the loaded value

              out           OCR0, mpr            ; Place for timer 0
              out           OCR2, mpr            ; Place for timer 2


              ret

SPEED_MIN:
              ldi           mpr, $00             ; Loaded minimum value into mpr
              out           OCR0, mpr            ; Output into Output compares
              out           OCR2, mpr


              ret

SPEED_MAX:
              ldi           mpr, $FF             ; Loaded maximum value into mpr
              out           OCR0, mpr            ; Output into Output compares
              out           OCR2, mpr


              ret

WAIT:
```

```
                push    r18                     ; Save wait register
                push    r19                     ; Save ilcnt register
                push    r20                     ; Save olcnt register

                ldi             r18, 200
Loop:   ldi             r20, 224                ; load olcnt register
OLoop:  ldi             r19, 237                ; load ilcnt register
ILoop:  dec             r19                     ; decrement ilcnt
                brne    ILoop                   ; Continue Inner Loop
                dec             r20             ; decrement olcnt
                brne    OLoop                   ; Continue Outer Loop
                dec             r18             ; Decrement wait
                brne    Loop                    ; Continue Wait loop

                pop             r20             ; Restore olcnt register
                pop             r19             ; Restore ilcnt register
                pop             r18             ; Restore wait register
                ret


WAIT_2:
                ldi             mpr, high(52000)        ; Load equation value
                out             TCNT1H, mpr
                ldi             mpr, low(52000)                 ; to set time for wait
                out             TCNT1L, mpr
LOOPY:
                in              mpr, TIFR                       ; Check TOV1 for overflow
                sbrs    mpr, TOV1
                rjmp    LOOPY
                ldi             mpr, 0b00000100                 ; restore flags
                out             TIFR, mpr
```

```
                ret


;*************************************************************
;*      Stored Program Data
;*************************************************************
                ; Enter any stored data you might need here


;*************************************************************
;*      Additional Program Includes
;*************************************************************
                ; There are no additional file includes for this program
```